FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Otimização do Parqueamento de Aeronaves em Hangares de Manutenção

Bruno Manuel João Estevinho

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: José Fernando da Costa Oliveira

31 de Janeiro de 2018

Resumo

O mercado da aviação é um dos mais importantes e com maior volume no transporte de pessoas e mercadorias. De maneira a fornecer um serviço com a melhor qualidade e os menores custos possíveis, as companhias aéreas necessitam de executar manutenções frequentes aos aviões que dispõem. Este documento pretende estudar o parqueamento de aeronaves em hangares, mais especificamente nos de prestadores independentes do serviço de manutenção, reparação e inspeção (MRO). Devido ao crescimento deste mercado, a procura de prestadores de serviço externos às companhias aéreas aumentou significativamente.

O problema a discutir é de otimização combinatória e enquadra-se na classe de problemas de Cortes e Empacotamento. Neste mercado em crescimento e cada vez mais competitivo, é imperativo responder às necessidades dos clientes com rapidez e eficácia, maximizando os lucros dos prestadores do serviço de MRO. Assim, perante um conjunto de pedidos de manutenção que excedam a capacidade máxima do hangar, o prestador do serviço tem de escolher um subconjunto de aviões para atender primeiramente, de maneira a maximizar o lucro. Deve repetir este processo até responder a todos os pedidos em espera. É necessário então ter em consideração que não podem haver sobreposições entre aviões e que devem haver margens de segurança mínimas entre eles.

Foi desenvolvida uma ferramenta de maneira a otimizar o processo descrito. Recorreu-se à construção de Nofit Polygon's (NFP) para assegurar estes constrangimentos e a métodos de ordenação para a colocação das aeronaves por forma a minimizar a ocupação do hangar. As simulações realizadas são baseadas em dados reais de uma companhia de manutenção de aviões e os resultados são validados computacionalmente através do programa desenvolvido.

Abstract

The aviation market is one of the most important and largest in the transport of people and goods. In order to provide a service with the best quality and the lowest possible costs, airline companies need to perform frequent maintenance on the planes they have. This document intends to study the parking of aircrafts in hangars, more specifically those of independent maintenance, repair and overhaul (MRO) service providers. Due to the growth of this market, the demand for external service providers has increased significantly.

The problem to be discussed is of combinatorial optimization and falls into the class of Strip Packing problems. In this growing and increasingly competitive market, it is imperative to respond to customer needs quickly and effectively, maximizing the profits of MRO service providers. Thus, faced with a set of maintenance requests that exceed the maximum capacity of the hangar, the service provider must choose a subset of aircraft to serve first, in order to maximize profit. You must repeat this process until all waiting requests are served. It is necessary to take into account that there can be no overlap between aircraft and that there should be minimum safety margins between them.

A tool has been developed in a way to optimize the process described. The construction of Nofit Polygon's (NFP) will be used to ensure these constraints and sorting methods for the placement of aircrafts in order to minimize the occupation of the hangar. The simulations made are based on actual data from an aircraft maintenance company and the results are computationally validated through the developed program.

Agradecimentos

Esta dissertação marca a conclusão de uma etapa fundamental da minha vida académica e pessoal. Ao longo destes anos foram inúmeros os obstáculos e muitos os que de forma direta ou indireta me ajudaram a ultrapassá-los. Quero agradecer a algumas pessoas em particular:

Ao Professor José Fernando Oliveira e à Professora Maria Antónia Carravilla, por tudo o que me ensinaram, por me ajudarem na compreensão e desenvolvimento do trabalho, toda a disponibilidade e interesse, a motivação dada em cada reunião e acima de tudo, a simpatia e amabilidade demonstrada.

Aos amigos que levei para a faculdade, os que me acompanharam durante o percurso académico, aos que levo para fora e todos os outros que fazem de mim a pessoa que sou.

Por fim, mas não menos significante, quero agradecer aos meus pais, ao meu irmão e à minha namorada pelo apoio incondicional e por ajudarem na minha evolução académica, profissional e acima de tudo pessoal.

Bruno Manuel João Estevinho

"Real knowledge is to know the extent of one's ignorance."

Confucius

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

MRO	Maintenance, Repair and Overhaul
NFP	Nofit Polygon
det	Determinante de uma matriz
SVG	Scalable Vector Graphics

Capítulo 1

Introdução

1.1 Enquadramento

Tal como observado por Keeling (2007: 220), o transporte encontra-se no núcleo da interação global. Uma das indústrias de transporte, que é um dos grandes propulsores de crescimento e desenvolvimento, é a da aviação. O transporte aéreo é hoje em dia, um dos principais meios responsáveis pelo transporte de pessoas e mercadorias, fortalecendo o contacto entre diferentes regiões do mundo e as atividades económicas entre países. Em Junho de 2017, a *International Air Transport Association* (IATA) divulgou no seu relatório semi-anual de performance económica da industria da aviação, dados que consolidam a tendência que se tem verificado ao longo dos últimos anos, que afirmam este mercado como um dos mais indispensáveis ao desenvolvimento económico e à globalização.(Pearce, 2017) Com o crescimento gradual deste mercado, acresce também o investimento na tecnologia das aeronaves, a importância da manutenção das mesmas e os custos totais associados.

Uma reflexão sobre os estudos existentes relativos a manutenção de aviões, é descrita em detalhe por Van den Bergh et al. (2013). O facto de ser muitas vezes necessário recorrer a múltiplos locais para o serviço de MRO é um dos tópicos abordados. A grande maioria dos estudos sobre problemas de manutenção de aviões, considera o uso de múltiplas bases no que toca a companhias comerciais, enquanto o uso de apenas uma se enquadra normalmente em aplicações militares. Torna-se cada vez mais fundamental para as companhias aéreas proporcionar o melhor serviço possível maximizando em simultâneo os seus lucros. Desta maneira, é necessário recorrer frequentemente a prestadores de serviços de manutenção externos. Estas empresas têm o desafio de lidar diariamente com aeronaves de diferentes proporções nos seus hangares, contrariamente ao que se verifica em cada companhia aérea que detém por hábito hangares adaptados às dimensões, no geral homogéneas, da sua frota.

1.2 Objetivos

Nesta dissertação apresenta-se um algoritmo original para o problema de alocação de estacionamento de aeronaves em hangares, especificamente para o caso dos prestadores independentes do serviço de manutenção, reparação e inspeção (MRO). Dado um conjunto de pedidos de manutenção que excedam a capacidade máxima do hangar, o objetivo é que seja escolhido um subconjunto de aviões para atender primeiramente de modo a maximizar o lucro, e posicionar os aviões no hangar de acordo com as restrições técnicas especificas do problema. O processo é repetido de maneira a executar todos os pedidos em espera. A Fig.1.1 exemplifica o resultado de uma colocação de um conjunto de aeronaves.

Para tal, é imperativo a inexistência de sobreposições entre aviões e a consideração de margens de segurança mínimas entre eles. Pretende-se um estudo ao nível das ferramentas geométricas disponíveis para assegurar estes constrangimentos e a métodos de ordenação para a colocação das aeronaves.

Por fim, a partir de dados reais de uma companhia de manutenção, são executadas as simulações na ferramenta desenvolvida e, com base nos resultados obtidos, efetuadas comparações com estudos efetuados previamente sobre a abordagem a este problema.

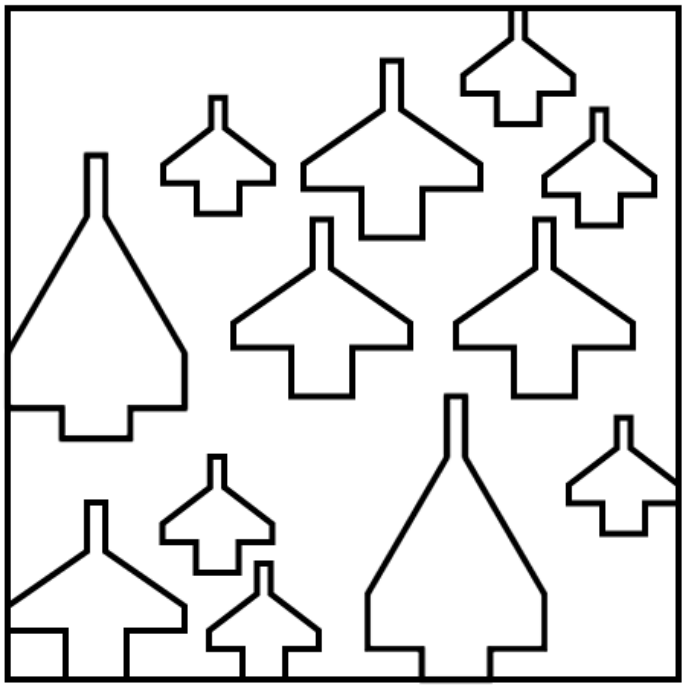

Figura 1.1: Conjunto de aviões colocados no hangar com as respetivas margens de segurança

1.3 Estrutura da Dissertação

A composição deste documento foi disposta por 5 capítulos, sendo este o primeiro. O propósito da estruturação realizada é de proporcionar uma visão detalhada, porém clara, do tema abordado bem como do trabalho desenvolvido.

É então feita no capítulo 2, uma abordagem ao estado da arte relacionada fundamentalmente com as ferramentas geométricas usadas na resolução de problemas de empacotamento de formas irregulares. São também detalhados os métodos escolhidos para o desenvolvimento do trabalho, bem como as assunções feitas e devidas justificações.

Seguidamente, é descrito no capítulo 3, todo o processo de implementação da ferramenta desenvolvida para a colocação de um dado número de aviões num hangar de manutenção.

As simulações realizadas são descritas no 4. São detalhados todos os dados usados e resultados obtidos consequentemente.

No Capítulo 5 é feita uma conclusão com base nos resultados obtidos e restante desenvolvimento desta dissertação. São também apresentadas propostas de trabalho futuro na área de conhecimento estudada.

Capítulo 2

Estado da Arte

2.1 Introdução

O problema desenvolvido nesta dissertação está inserido nas áreas da geometria na colocação de objetos e dos problemas de empacotamento de formas irregulares. Os estudos sobre estas áreas são vastos, assim como as abordagens disponíveis para os solucionar. Primeiramente foi elaborada uma revisão literária sobre o que existe acerca de ferramentas geométricas que permitam a deteção de colisões de objetos e mais tarde sobre os métodos de colocação de formas irregulares, comummente chamados de problemas de nesting.

2.2 Nofit Polygon

Quando se resolvem problemas de *nesting*, uma das principais características é a necessidade de técnicas para lidar com geometria. Uma das mais populares é o *nofit polygon* (NFP), em grande parte pela eficiência relativamente à trigonometria direta. No geral, a sua utilização é fundamental para determinar se dois polígonos se sobrepõem, tocam ou estão separados. É bastante vantajosa a aplicação desta ferramenta pela sua eficiência computacional, porém é complicado desenvolver um gerador de NFP's robusto para polígonos não convexos.

O NFP entre dois polígonos A e B é o resultado do deslizamento de um sobre o outro, em que ambos têm uma orientação fixa. A denotação NFP_{AB}, refere-se ao polígono produzido por A fixo e B deslizando à volta do seu perímetro, sendo B o polígono que traça o NFP. Isto é, B é colocado numa posição em que toca A e um ponto de referência de B delimita o novo polígono à medida que este desliza sobre A, sem nunca o sobrepor nem deixar de tocar (Figura 2.1a).

Ao trocar o polígono fixo com o que desliza, o NFP obtido será o anterior girado 180 graus (Figura 2.1b). O interior do NFP representa toda a área dentro da qual o ponto de referência usado pelo polígono deslizante não se deve encontrar, para garantir que não há sobreposição de polígonos.

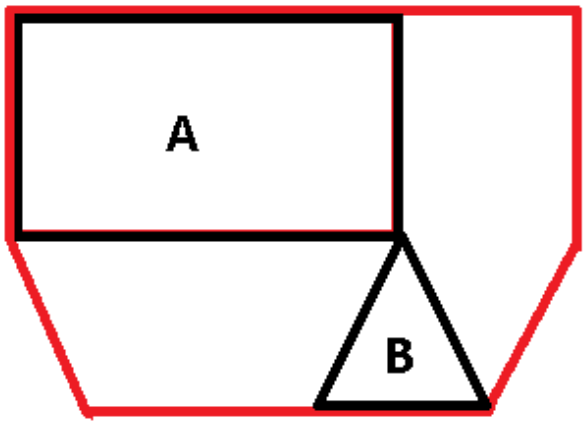

Figura 2.1: Exemplo de desenho de NFP

2.2.1 Formas convexas e não-convexas

Como é explicado por Cuninghame-Green (1989), a forma mais simples para gerar NFP's ocorre quando ambos os polígonos são convexos. Neste caso, o cálculo do NFP de duas formas convexas A e B, é criado seguindo os seguintes passos:

- definir a orientação de A como anti-horária e B horária,
- todas as arestas de A e B são representadas a partir de um ponto singular,
- ligar as arestas por ordem anti-horária para obter o NFP final.

Contudo, quando o processo envolve formas não-convexas, o cálculo torna-se mais complexo. De seguida são apresentados os três métodos mais usuais na produção de NFP's a partir de formas não-convexas.

2.2.2 Somas de Minkowski

Este tipo de operação envolve uma técnica para gerar o NFP chamada de dilatação. Basicamente, a dilatação faz crescer o tamanho de um polígono A através de adições vetoriais com outro polígono B.

Formalizado pela primeira vez por Stoyan e Ponomarenko (1977), o conceito define que dados dois conjuntos fechados de pontos vetoriais A e B em $R\hat{2}$, S é a soma resultante da adição de todos os pontos vetoriais em A com os de B. A soma de Minkowski, S, é definida por:

$$S = A \oplus B = \{a + b \,|\, a \in A, b \in B\} \tag{2.1}$$

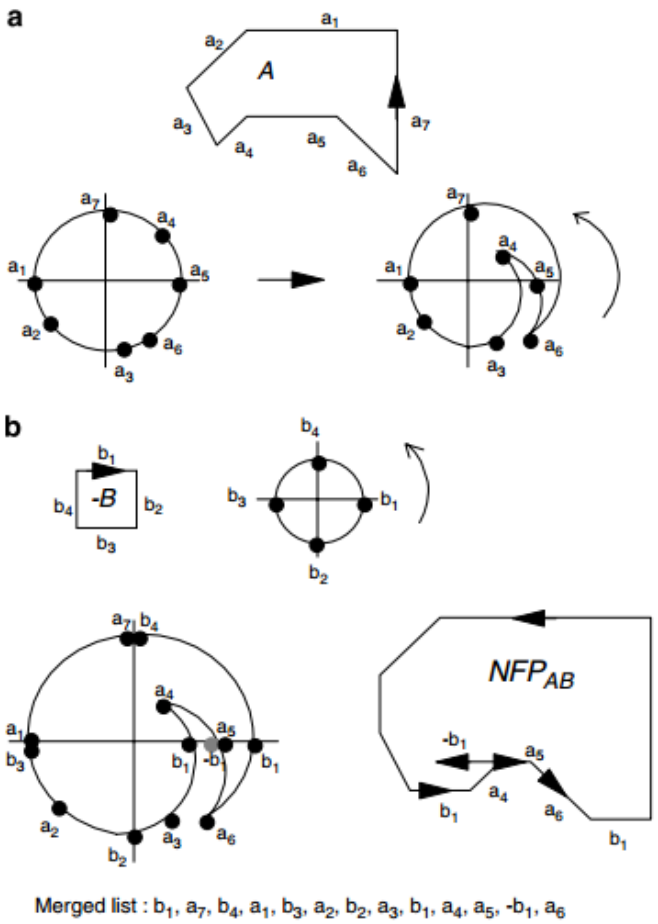

Figura 2.2: Calculo do NFP entre um polígono côncavo e um convexo (Bennell e Oliveira (2008))

Para a construção de um NFP a partir deste método, é necessário assumir que ambos os polígonos A e B têm a mesma orientação. Usa-se a diferença de Minkowski, A $\oplus$ -B para realizar o cálculo com o conjunto simétrico de B (Bennell et al. (2000)).

Por forma que se calcule as somas de Minkowski e a sua relação com os NFP, a definição acima descrita deve ser traduzida num procedimento para a obtenção do NFP. Ghosh (1991) desenvolveu um conjunto de teoremas para os casos convexo e não convexo que sustentam o seu método de obtenção das somas de Minkowski. Estes teoremas baseiam-se nos limites dos polígonos para obter os limites das somas de Minkowski, sendo que estes são, na visão do autor, uma forma mais concisa de representação. Visto que o teorema é definido pelos conjuntos de pontos dos polígonos, foram necessárias alterações para considerar em alternativa os limites dos polígonos. O teorema suporta o uso dos diagramas de declive que formam a base da sua abordagem para representar as somas de Minkowski. Bennell e Oliveira (2008) demonstram facilmente através de um exemplo a abordagem de Ghosh (Fig.2.2).

O exemplo apresenta um polígono com uma concavidade e um polígono B convexo. Para facilmente se entender e executar o calculo é representado o diagrama de declives para cada polí-

gono, e o diagrama final resultante da junção para o NFP. Cada aresta é desenhada no diagrama de acordo com o seu declive e orientação por ordem. Assume-se a orientação de B em oposição à de A e consequentemente, as arestas aparecem por ordem inversa ao examinar o diagrama na direção anti-horária. Observa-se que o facto de B ser convexo resulta no desenho das arestas em sequencia, o que não acontece em A por ser não-convexo. Como tal, o diagrama de A é redesenhado para que as arestas se encontrem por ordem, deixando de apresentar a forma circular.

Para obter o NFP, os diagramas de A e B são reunidos por forma a obter um só mantendo as ordens dos declives. Os declives sobrepostos no diagrama são somados para obter a sua representação no diagrama do NFP. Tomando como exemplo a Fig.2.2, a1 e b3 têm o mesmo declive, sendo por isso marcados em conjunto no diagrama do NFP. Imediatamente, seguindo a direção horária, é marcado o declive a2. O processo segue a mesma lógica mantendo sempre a ordem das arestas representadas. De salientar que b1 é atravessado três vezes, pelo que esta aresta é representada uma vez mais mas com sinal negativo e a terceira vez ao passar entre a6, a7 e b4.

O NFP resultante não é um simples polígono pois apresenta arestas internas. Esta particularidade advém do teorema de adição de limites, como explicado com mais detalhe por Ghosh (1991). Estas arestas ou *loops* possíveis de aparecer no resultado final, necessitam de ser verificados pois podem representar buracos (Bennell e Song (2008)).

Desde que as concavidades não interfiram umas com as outras, o método explicado pode ser usado nos casos em que ambos os polígonos são não-convexos. Porém, caso se verifique o contrário, vão existir partes do diagrama de declives em que ambos os conjuntos de arestas não se encontram por ordem sequencial. Ghosh (1991) lida com este pormenor separando o diagrama de declives em dois caminhos, um para cada concavidade que interage, onde cada caminho através do diagrama de declives define um polígono. A união ou face exterior destes polígonos forma assim o NFP.

Bennell e Song (2008) apresentam uma nova abordagem baseada no teorema de adição de limites. A vantagem deste método encontra-se na sua simplicidade e elimina a ambiguidade no que concerne a quais arestas devem ser consideradas. Mantendo um polígono A como inalterado, a ideia é partir o polígono B em grupos de arestas sequenciais no sentido horário ou anti-horário. Cada um destes grupos pode depois ser individualmente junto com o diagrama de declives de A sem conflito. Ao combinar a lista de arestas, são necessárias arestas de ligação por forma a manter a precedência de arestas de cada polígono. Como os grupos criados não formam um ciclo, consideram-se as arestas como uma lista e define-se a aresta inicial como sendo a primeira de B no grupo. Sendo que os grupos se encontram ligados, cada grupo é completado atravessando na direção anti-horária. Na Fig.2.3 é dado o exemplo usado por Bennell e Song (2008).

Começando a partir da aresta b4, procura-se a seguinte aresta, b5. Para a encontrar, a3 e a4 são atravessados. Partindo de b5 procura-se b1 atravessando a5. De b1 segue-se para b2 passando por a1 e finalmente para b3 atravessando a2. A aresta b3 vai aparecer três vezes com direção negativa quando encontrada ao percorrer o diagrama de declives no sentido horário, devido à concavidade de A. Até serem encontradas todas as ocorrências de b3, a procura continua. Como um polígono equivale a um ciclo completo, o fim da lista é ligado ao inicio. Partindo de b3 procura-se b4, o

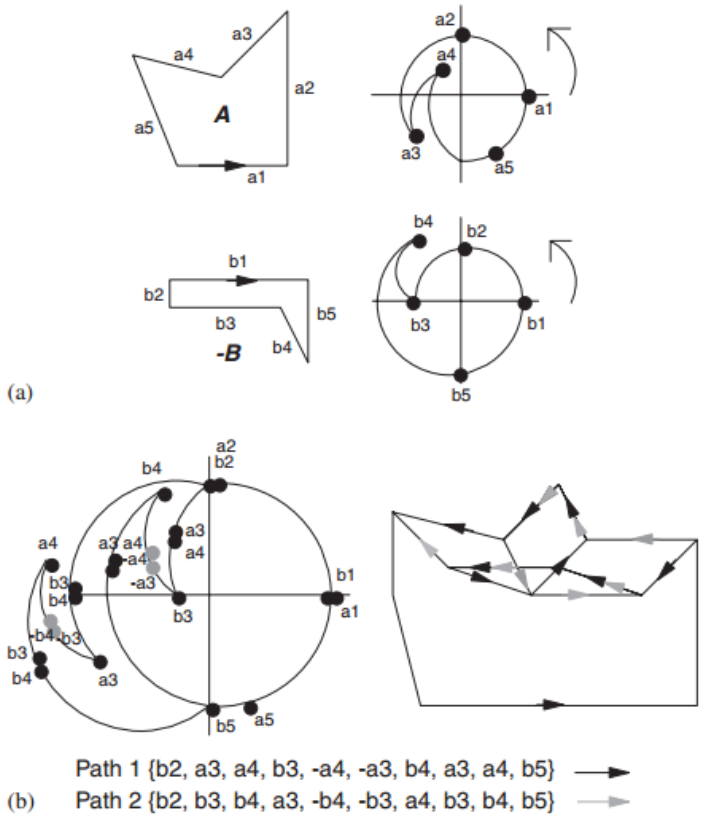

Figura 2.3: Interação de concavidades entre dois polígonos (Bennell e Song (2008))

que requer uma volta ao diagrama de declives no sentido horário, e atravessando -a4 e -a3. A lista final obtida é: b4,a3,a4,b5,a5,b1,a1,b2,a2,b3,a3,-b3,a4,b3,-a4,-a3.

2.2.3 Decomposição

Uma das formas para facilitar o cálculo do NFP quando existem concavidades de polígonos em interação é através da decomposição das formas em peças mais fáceis de manipular. Agarwal et al. (2002) executam a decomposição de polígonos simples em peças convexas, como é explicado mais à frente. Porém a desvantagem encontra-se nos NFP's resultantes de cada sub-peça criada. Para posteriormente se executar o teste de interseções, os NFP's gerados podem manter-se decompostos ou ser recombinados. Quando se executa a recombinação das peças decompostas, um único NFP oferece uma computação das interseções mais rápida mas requer cálculos adicionais para obter a recombinações das partes constituintes.As grandes desvantagens associadas a esta abordagem são os elevados custos computacionais e a dificuldade acrescida caso se trate de várias sub-peças e na presença de buracos.

Alguns algoritmos para decomposição de polígonos em sub-peças convexas são sugeridos por Fernández et al. (2000). A vantagem destes centra-se no facto de não ser necessária a adição de

novos vértices para a execução. No seu estudo, concluem que o uso de processos de recombinação é recomendado por ser rápido e reduzir consideravelmente o número de polígonos. Porém as abordagens descritas não incluem a consideração de polígonos com buracos que são um problema bastante recorrente. Agarwal et al. (2002) analisam os diferentes algoritmos de decomposições e operações de recombinação das sub-partes para construções eficientes das somas de *Minkowski* de polígonos não-convexos. Eles concluem que é contra-produtivo o uso de decomposições ótimas, pois os tempos de computação para as calcular sobrepõem-se aos benefícios atingidos durante a recombinação, e que a otimização mais eficiente passa por minimizar o número de sub-polígonos convexos. Os autores afirmam que a operação de recombinação acarreta grande custos e dão exemplos de tempos de execução de vão de alguns segundos para formas com pequenas quantidades de concavidades até vinte minutos para formas altamente irregulares. Avnaim e Bsissonnat (1987) apresentam uma abordagem de recombinação de decomposições baseada em segmentos lineares. Estes são usados para produzir paralelogramos que são recombinados para produzir o NFP. Os autores apresentam fundamentos matemáticos que mostram que a abordagem pode suportar o caso geral e extender para permitir a rotação. De seguida são expostas as abordagens mais comuns para a decomposição dos polígonos.

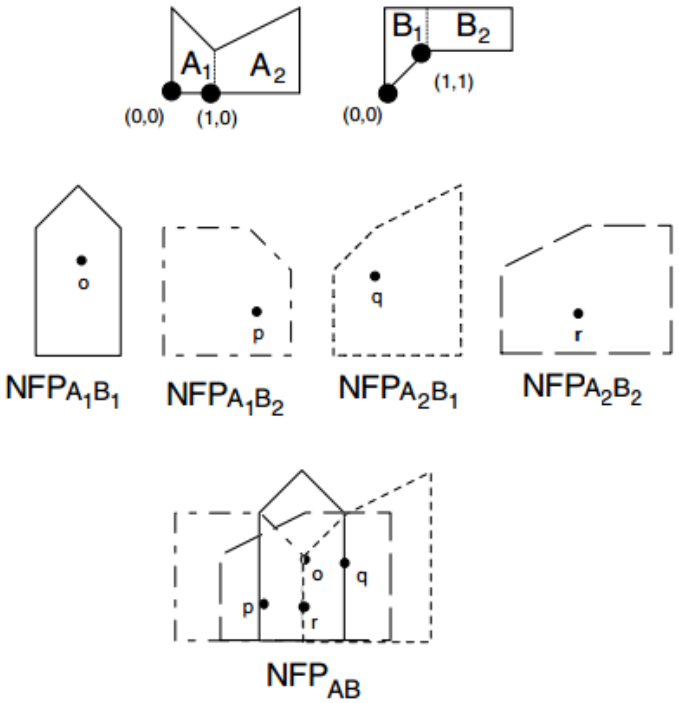

Figura 2.4: Exemplo de Bennell e Oliveira (2008) para o cálculo de um NFP através da decomposição em formas convexas

2.2.3.1 Peças convexas

Tal como discutido anteriormente, a produção do NFP é bastante fácil quando é entre polígonos convexos. Se uma forma com concavidades pode ser dividida em peças convexas, técnicas rápidas de produção de NFP podem ser utilizadas. A abordagem à decomposição de formas não-convexas para peças convexas relativa ao problema das interseções geométricas foi discutida em Cuninghame-Green (1992). O autor expõe que a maneira mais simples de decomposição é executada ao traçar as divisões a partir dos vértices do polígono não-convexo que apontam para o interior deste, até encontrar uma aresta. Se o polígono em questão apresentar n vértices que apontam para o interior, o processo de decomposição resulta em n+1 sub-polígonos. Por outro lado, desta forma não são produzidos o menor número de peças necessário. As principais dificuldades nesta abordagem são, para além da decomposição eficiente em si, os algoritmos de recombinação. A abordagem da decomposição em peças de forma triangular realizada por Seidel (2010) produz mais sub-peças do que é realmente necessário e vai ter um grande impacto nos tempos de computação da produção do NFP. Ao contrário da triangulação, o algoritmo de decomposição em formas convexas pretende representar polígonos com o menor numero de peças convexas.

A Fig.2.4 exemplifica o método de produção do NFP entre dois polígonos, a partir da decomposição destes em formas convexas.

2.2.3.2 Polígonos em forma de estrela

Li e Milenkovic (1995) decompõem polígonos em sub-peças em forma de estrela. Eles descrevem que para um polígono P, existe um *kernel point k* no seu interior que consegue ver todos os limites do polígono. Para um polígono convexo, o kernel (conjunto de kernel points) equivale ao polígono mais o seu interior. No caso de polígonos simples, o kernel pode ou não encontrar-se vazio. Se se verificar que este não está vazio, então estamos na presença de um polígono em forma de estrela.

Através da extensão das arestas das concavidades e eliminação de regiões invisíveis, é evidente que o polígono em forma de estrela tem uma região que pode ser definida na qual um ponto kernel pode ser colocado para ver os limites de todo o polígono. Por outro lado, isto não acontece num polígono que não tenha a forma em estrela pois não existe região que possa ser definida para colocar o ponto kernel.

Assim, polígonos em forma de estrela estão situados algures entre formas convexas e não convexas em termos de generalidade. Li e Milenkovic (1995) afirmam que polígonos em forma de estrela são fechados sob operações de somas de Minkowski e fornecem uma prova de que a soma de Minkowski entre dois polígonos em forma de estrela também produz um polígono em forma de estrela. Os autores não afirmam se recombinam as regiões do NFP numa só entidade ou se realizam múltiplos testes de interseções de NFP durante a produção do *layout*.

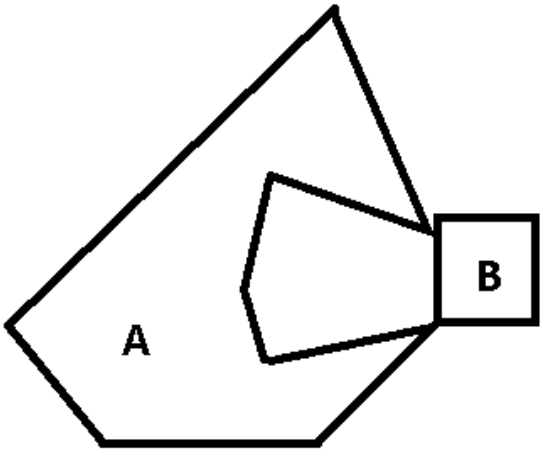

Figura 2.5: Poligono B desliza sobre a conavidade de A

2.2.4 Algoritmo Sliding

A abordagem pelo algoritmo Sliding, envolve o uso de trigonometria simples para efetuar o deslizamento de um polígono em redor do outro. A primeira abordagem ao método é descrita em Mahadevan (1984). Os elementos chave da abordagem de Mahadevan são: cálculo dos vértices e arestas em contacto, determinação do vetor de translação e cálculo da distância de translação. O cálculo das interseções é realizado através do uso das funções D (Konopasek (1981)). Mahadevan modifica o teste das funções D para calcular também pontos de contacto, que é necessário tanto para o algoritmo de Mahadevan como a nova abordagem por Burke et al. (2007). Esta informação é depois usada para selecionar um vetor de translação com base na aresta em contacto. O vetor de translação é então projetado a partir de cada vértice do polígono em orbita e depois o teste da aresta intersectante é usado para calcular a distância de translação. É também importante projetar o vetor de translação na direção inversa do polígono fixo.

Na forma em orbita é aplicada depois a translação ao longo do vetor de translação pela menor distância (dos testes de projeção e interseção). Assim é assegurado que os dois polígonos nunca se sobrepõem mas sempre se tocam. O processo continua até o polígono em orbita retornar à sua posição inicial.

A maior desvantagem presente no algoritmo de Mahadevan é a impossibilidade de gerar um NFP completo para formas que apresentem buracos ou algumas concavidades. Mais precisamente, o problema ocorre quando o polígono que orbita consegue ser colocado dentro de uma concavidade do polígono fixo mas a concavidade tem uma entrada estreita. Nestas circunstâncias, o polígono orbitante é largo demais e irá deslizar sobre a concavidade. A figura 2.5 demonstra este caso.

Burke et al. (2007) sugerem então uma abordagem atualizada para superar este problema. Esta é dividida em duas partes. Na primeira é criado o desenho do perímetro do NFP entre duas formas seguindo uma lógica similar à do algoritmo de Mahadevan, embora uma implementação

modificada seja proposta. Na segunda parte é introduzida a noção e identificação de posições iniciais que permitem ao algoritmo encontrar os caminhos restantes. Estes caminhos serão buracos contidos no NFP que não são encontrados usando apenas o algoritmo de Mahadevan. Assume-se nesta secção que se tem dois polígonos, A e B, orientados no sentido anti-horário e que existem num espaço bidimensional.

Para produzir o NFP_{AB} (polígono orbitante B à volta do polígono A), a primeira operação a ser realizada é a translação do polígono B para que este se encontre em contacto, sem interseção, com o polígono A. Mantém-se o processo usado por Mahadevan em que o polígono B é transladado de maneira que a sua maior coordenada y é posicionada na menor coordenada y de A.

Usando estes dois vértices, A e B tocam-se e não estarão em interseção. A translação que resulta no polígono B em contacto com A é descrita na formula 2.2:

$$\text{Trans } B \rightarrow A = Pt_{A(ymin)} - Pt_{B(ymax)} \tag{2.2}$$

Na verdade, qualquer posição inicial pode ser usada desde que se verifique que os polígonos A e B se tocam e não se intersetam. Inicia-se então o desenho do perímetro do NFP na direção anti-horário.

O principal objetivo da parte do deslizamento do algoritmo é a deteção dos movimentos corretos que B tem de efetuar em redor de A de maneira a retornar à posição de partida. Trata-se de um procedimento iterativo em que cada translação cria uma aresta do NFP. Este processo pode ser decomposto nas seguintes sub partes a discutir de seguida: deteção de arestas em contacto, criação de potenciais vetores de translação, encontrar uma translação viável, aparar a translação viável e finalmente, aplicar a translação viável.

A habilidade de detetar corretamente arestas em contacto ou que se intersetem é primordial. É atingida através do teste de cada aresta do polígono A contra cada aresta de B. Cada par de arestas em contacto é guardado juntamente com a posição do vértice em que se tocam.

O vetor com que o polígono B será transladado para orbitar em redor de A deriva de uma aresta de A ou de B, dependendo da situação. De notar que quando se verifica a situação de B deslizar ao longo da própria aresta, o vetor de translação é encontrado invertendo a aresta deste, visto que o polígono A se deve manter fixo.

Assim que todos os potenciais vetores de translação forem produzidos, seleciona-se de seguida um vetor de translação que não resulte numa interseção imediata.

Por exemplo, na Fig.(2.6) dois potenciais vetores de translação, a1 e b3. Pode-se observar que o polígono orbitante B tem de mover-se ao longo do vetor b3 de maneira a deslizar no sentido anti-horário à volta de A, sem que resulte numa colisão.

Para que um potencial vetor de translação seja identificado como viável, ele deve ser válido para cada par de arestas tocantes. Quando um potencial vetor de translação falha na verificação, a translação é inviável e por consequente eliminada.

O ultimo passo antes da translação de B, é aparar o vetor de translação. Isto é importante devido a arestas que interfiram com o deslocamento do polígono orbitante, ou seja, que causem

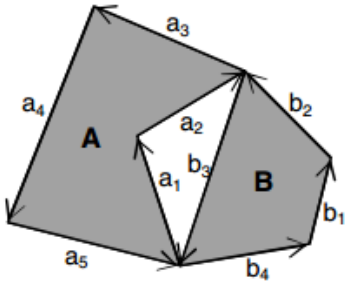

Figura 2.6: Dois polígonos em contacto em duas posições diferentes (Burke et al. (2007))

interseção entre as formas.

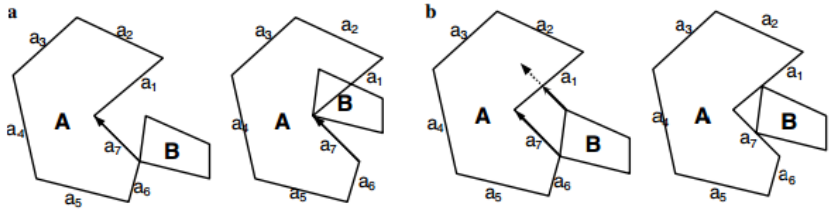

Figura 2.7: Um vetor que necessita de ser aparado para evitar uma interseção (Burke et al. (2007))

A segunda parte da abordagem de Burke et al. (2007) baseia-se na identificação de pontos iniciais válidos, em contacto e não intersetantes, com os quais a técnica de sliding previamente descrita pode ser aplicada para gerar o restante perímetro interior do NFP derivado dos buracos dos polígonos.

Se for possível colocar o polígono B no interior do buraco de A, então o algoritmo de deslizamento pode ser aplicado para gerar a região interior do NFP. A figura 2.8 exemplifique a procura de um ponto inicial válido para a colocação de de um polígono dentro de uma concavidade de outro.

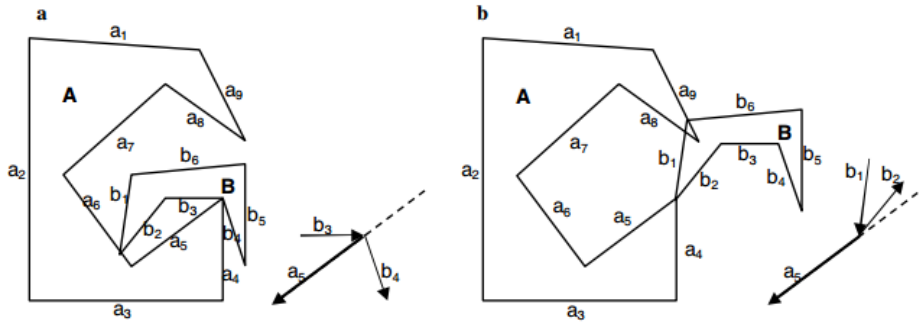

Figura 2.8: Caso (a) ilustra o posicionamento do polígono B dentro de A; Colisão de polígono B com A por posicionamento inválido (b) (Burke et al. (2007))

O método é o utilizado anteriormente, com a particularidade de que o interesse nesta parte é o de resolver interseções de arestas enquanto se move B ao longo de um vetor em particular.

Resumindo, o algoritmo calcula o perímetro do NFP exterior usando a abordagem descrita e depois aplica o procedimento de procura de pontos iniciais nas arestas por marcar como verificadas tanto de A como B. Assim que um ponto inicial válido é detetado, é calculada a curva interior do NFP, marcando as arestas como anteriormente à medida que são percorridas. O procedimento repete-se até todas as arestas serem verificadas e marcadas como tal.

2.2.4.1 Funções Phi

Uma recente e promissora abordagem aos problemas geométricos é apresentada por Stoyan et al. As funções phi são baseadas nas funções de distância entre dois objetos e assim sendo podem ser facilmente usadas para descobrir a distância entre dois objetos complexos. Algumas das grandes vantagens associadas a este método são a precisão na modelação de problemas de colocação, a possibilidade de generalizar problemas de objetos tridimensionais e consideração de rotação.

Embora não sejam uma abordagem totalmente focada na produção de NFP's, sendo apenas um caso específico, revela resultados muito promissores. Os autores desenvolvem ainda mais o seu trabalho em Stoyan et al. (2004), para permitir a definição matemática de relações de interseções para polígonos não-convexos através da união, interseção e complemento de objetos primários. O teste das interseções resultantes entre dois polígonos é realizado através de comparações das funções *phi* entre os pares de objetos primários que definem uma forma A e uma forma B.

A grande desvantagem deste método, e razão de não ser usada mais frequentemente, deve-se ao facto de não existir um algoritmo para gerar as funções *phi* para formas aleatórias.

2.3 Resumo

Todos os métodos e abordagens consideradas têm as suas vantagens e desvantagens. É importante ter em consideração o problema exposto de maneira a escolher um que suporte o desenvolvimento e resolução dos objetivos pretendidos de forma eficaz e eficiente, essencialmente a nível computacional. Excetuando o caso em que os polígonos são convexos, todos estes métodos têm uma dificuldade acrescida numa implementação robusta dos mesmos. Devido a isto, a abordagem desenvolvida procura tirar partido da geometria particular dos aviões para gerar os NFP's de maneira totalmente robusta e de fácil implementação, apesar de ser apenas aplicável a formas deste tipo.

Capítulo 3

Desenvolvimento

O módulo base do algoritmo de posicionamento dos aviões no hangar é o que garante que os aviões não se sobrepõem. Para tal recorreu-se ao conceito de NFP. Em vez de recorrer a algoritmos genéricos de geração de NFP's, descritos no capitulo anterior 2 (que têm problemas de robustez), desenvolveu-se um algoritmo original que, tirando partido da geometria dos aviões, permite uma implementação totalmente robusta e imune aos problemas de precisão numérica que afetam os anteriores métodos. Note-se que o facto de a forma dos aviões ser não convexa transforma este algoritmo no único conhecido, com estas características, para polígonos não convexos, se bem que não seja diretamente extensível a polígonos genéricos.

O programa foi desenvolvido em linguagem C no editor de texto *SublimeText3* e compilado no *CodeBlocks*. De seguida é exposto o desenvolvimento do programa. O código é dividido em dois ficheiros para facilitar a execução em ciclo, como é explicado mais à frente (3.4).

No desenvolvimento do programa o sistema de coordenadas adotado tem o eixos das abcissas com valores positivos crescentes da esquerda para a direita e o das ordenadas com valores positivos crescentes no sentido de cima para baixo. Para a realização dos NFP's foi estabelecido que a origem de cada avião é dada pelo vértice com menor valor de y e menor x enquanto a origem do NFP é dada pelo maior valor de y e maior valor de x, como se pode observar na Fig.3.1.

3.1 Estudo dos NFP's dos Aviões

Antes do desenvolvimento da ferramenta pretendida, foi imperativo o estudo relativo ao NFP gerado por dois aviões. Assente na ideia de que era possível obter uma forma geral, pretendeu-se desenvolver um algoritmo capaz de devolver o NFP entre dois aviões, através de relações lineares simples e pré-determinadas executadas sobre as coordenadas destes.

Foi possível deduzir que existe uma relação entre as dimensões dos aviões e a forma final do NFP. A partir deste estudo foram escritas as relações geométricas que descrevem a forma geral do NFP entre dois aviões.

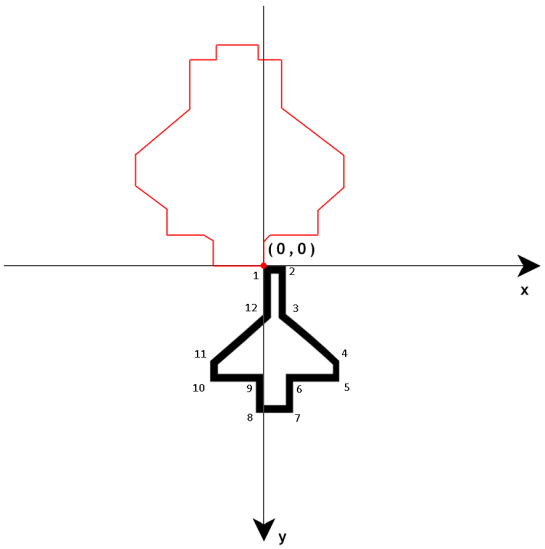

Figura 3.1: Origem de avião e NFP no sistema de coordenadas cartesiano

3.1.1 Função noFitAB

Com o intuito de facilitar o cálculo e obter um menor esforço computacional, foi criado de raiz um algoritmo que gera NFP's de forma fechada, ou seja, apenas para formas de aviões. A abordagem relaciona as coordenadas dos polígonos não convexos originais para obter o próximo vértice do NFP. Assim, para quaisquer formas de aviões que mantenham o número de vértices (seis) e sujeitas apenas a alterações nas suas dimensões, como se pode ver na Fig.(3.2), o algoritmo será capaz de calcular o NFP entre elas. O pseudocódigo 1 ilustra um excerto do cálculo do NFP desde o vértice 1 ao 16 (3.8), sendo que os vértices restantes são simétricos dos já obtidos. Há que ter em conta a origem dos aviões e dos NFP's (3.1).

Como se pode observar, o algoritmo implementado relaciona os vértices dos aviões para obter os vértices do NFP. A 3.3 ilustra o cálculo de um vértice do NFP. Neste exemplo, O avião B desliza sobre um vértice A e como não colide com A durante o movimento, o ponto seguinte do NFP é dado pela soma do ponto $NFP_{(x,y)}$ com a aresta de B que realiza o movimento.

3.1.2 InnerFit Polygon

O *innerfit polygon* é determinado através do desenho ao redor do perímetro interior do hangar, por um dos eixos do avião a colocar. Este retângulo obtido, representa toda a área na qual o avião pode ser colocado sem que haja colisão com o hangar. Na Fig.3.4 é exemplificado o desenho do *innerfit* (a vermelho), gerado pelo vértice P do avião ao deslizar sobre o interior do hangar (representado pelo retângulo preto). A área do *innerfit* obtida é toda a superfície onde o vértice P pode ser colocado garantindo que o avião não colide com o retângulo a negro.

Algorithm 1 Gerar NFP

function NOFITAB(A,B,NFP)
  $m \leftarrow (B_{12y} - B_{11y})/(B_{12x} - B_{11x})$;
  $m2 \leftarrow (A_{4y} - A_{1y})/(A_{4x} - A_{1x})$;
  $m3 \leftarrow (A_{4y} - A_{3y})/(A_{4x} - A_{3x})$;
  $NPF_{1x} \leftarrow A_{7x}$
  $NPF_{1y} \leftarrow A_{7y}$
  $NPF_{2x} \leftarrow A_{7x}$
  if $(B_{12y} - B_{1y}) \geq (A_{7y} - A_{6y})$ **then**
    $NPF_{2y} \leftarrow A_{6y}$
    $NPF_{3x} \leftarrow A_{7x}$
    $NPF_{3y} \leftarrow A_{6y}$
  else
    $NPF_{2y} \leftarrow (A_{7y} - (B_{12y} - B_{1y}))$
    $NPF_{3x} \leftarrow ((A_{5y} - NFP_{2y})/m) + A_{7x}$ ▷ Point given by the equation of a straight line
    $NPF_{3y} \leftarrow A_{6y}$
  end if
  $NPF_{4x} \leftarrow A_{5x}$
  $NPF_{4y} \leftarrow A_{5y}$
  $dist1 \leftarrow sqrt((A_{5x} - A_{7x}) * (A_{5x} - A_{7x}) + (A_{5y} - A_{7y}) * (A_{5y} - A_{7y}))$
  $dist2 \leftarrow sqrt((A_{5x} - A_{7x}) * (B_{12x} - B_{11x}) + (B_{12y} - B_{11y}) * (B_{12y} - B_{11y}))$
  if $(m2 < m) AND (B_{11y} - B_{12y} > A_{7y} - A_{6y})$ **then**
    $b \leftarrow NFP_{4y} - m * NFP_{4x}$
    $yaux \leftarrow m * NFP_{1x} + b$
    $yaux2 \leftarrow NFP_{1y} - yaux$
    $NPF_{5x} \leftarrow A_{5x}$
    $NPF_{5y} \leftarrow A_{5y} - (B_{12y} - B_{1y}) + yaux2$
    $NPF_{6x} \leftarrow NFP_{1x} + (B_{1x} - B_{11x})$
    $NPF_{6y} \leftarrow NFP_{1y} - (B_{11y} - B_{1y})$
    $NPF_{7x} \leftarrow NFP_{6x}$
    $NPF_{7y} \leftarrow NFP_{6y} - yaux2$
  else
    $NPF_{5x} \leftarrow A_{5x}$
    $NPF_{5y} \leftarrow A_{5y} - (B_{12y} - B_{1y})$
    $NPF_{6x} \leftarrow NFP_{5x}$
    $NPF_{6y} \leftarrow NFP_{5y}$
    $NPF_{7x} \leftarrow NFP_{6x}$
    $NPF_{7y} \leftarrow NFP_{6y}$
  end if
  $NPF_{8x} \leftarrow NFP_{4x} + (B_{12x} - B_{11x})$
  $NPF_{8y} \leftarrow NFP_{4y} - (B_{11y} - B_{1y})$
  $NPF_{9x} \leftarrow NFP_{8x}$
  $NPF_{9y} \leftarrow A_{4y} - (B_{10y} - B_{1y})$
end function

Algorithm 2 Gerar NFP (contin.)

function NOFITAB(A,B,NFP)
  if $(m2 > m3) AND (A_{4x} - A_{3x} > B_{9x} - B_{10x})$ **then**
    $b \leftarrow A_{4y} - m3 * A_{4x}$
    $yaux \leftarrow m3 * (A_{5x} - (B_{9x} - B_{10x})) + b$
    $yaux2 \leftarrow (B_{8y} - B_{9y}) - (A_{4y} - yaux)$
    $NPF_{11x} \leftarrow A_{4x} + (B_{1x} - B_{8x})$
    $NPF_{11y} \leftarrow A_{4y} - (B_{8y} - B_{1y})$
    $NPF_{10x} \leftarrow NFP_{11x}$
    $NPF_{10y} \leftarrow NFP_{11y} + yaux2$
    $NPF_{12x} \leftarrow A_{2x} + (B_{12x} - B_{11x})$
    $NPF_{12y} \leftarrow A_{3y} - (B_{10y} - B_{1y}) - yaux2$
  else
    $NPF_{10x} \leftarrow NPF_{9x}$
    $NPF_{10y} \leftarrow NPF_{9y}$
    $NPF_{11x} \leftarrow NFP_{9x}$
    $NPF_{11y} \leftarrow NFP_{9y}$
    $NPF_{12x} \leftarrow NFP_{9x} - (A_{4x} - A_{3x})$
    $NPF_{12y} \leftarrow NFP_{9y} - (A_{4y} - A_{3y})$
  end if
  $NPF_{13x} \leftarrow NFP_{12x}$
  $NPF_{13y} \leftarrow A_{2y} - (B_{10y} - B_{1y})$
  $NPF_{16x} \leftarrow NFP_{13x} - (B_{9x} - B_{10x})$
  $NPF_{16y} \leftarrow A_{2y} - (B_{8y} - B_{1y})$
  $NPF_{15x} \leftarrow NFP_{16x}$
  if $(B_{8y} - B_{9y}) \geq (A_{3y} - A_{2y})$ **then**
    $NPF_{15y} \leftarrow NFP_{16y} + (A_{3y} - A_{2y})$
    $NPF_{14x} \leftarrow ((NFP_{13y} - NFP_{15y})/-m) + NFP_{15x}$
    $NPF_{14y} \leftarrow NFP_{13y}$
  else
    $NPF_{15y} \leftarrow NPF_{13y}$
    $NPF_{14x} \leftarrow NFP_{16x}$
    $NPF_{14y} \leftarrow NFP_{13y}$
  end if
  .
  .
  .
end function

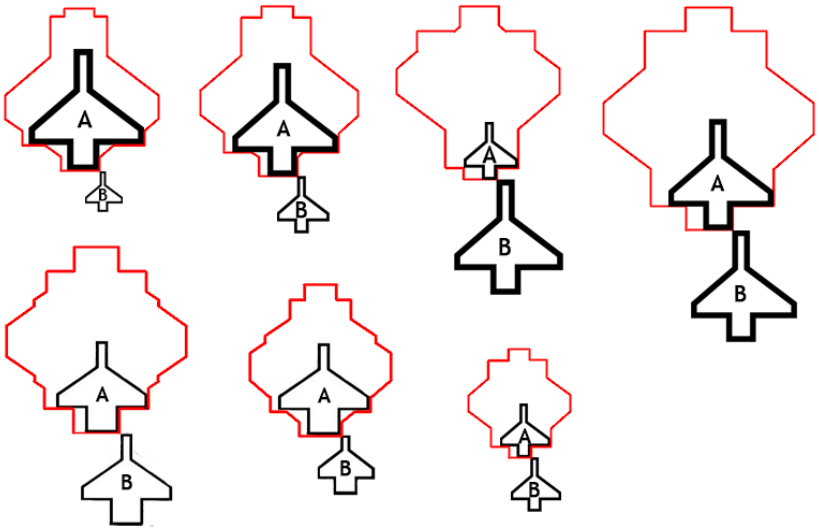

Figura 3.2: Estudo dos NFP's entre aviões de diferentes dimensões

3.2 Estrutura

O programa divide-se em duas partes principais. No geral, a primeira parte, ou biblioteca de rotinas de suporte, consiste na execução do cálculo dos NFP's dos aviões a colocar, a colocação dos mesmos e a produção de um ficheiro SVG para a sua visualização. A Fig.3.5, representa a estrutura geral.

A escolha de um ficheiro SVG como forma de visualização deve-se, em grande parte, à facilidade com que através das coordenadas podemos obter todas as formas geométricas a partir de um navegador Web. A segunda parte, Heurística de Colocação, é responsável por executar a colocação em ciclo ao receber um ficheiro de entrada com todos os aviões a colocar, as iterações pretendidas e o método de colocação escolhido. No final, obtém-se um ficheiro com os dados de cada ciclo, do qual é escolhida a melhor iteração e o respetivo resultado.

Algorithm 3 Desenho de innerfit polygon

```
1:  function INNERFIT(HANGAR, PLANE, INNER)
2:      inner[0](x) ← hangar[0] + distance from plane origin to end of left wing
3:      inner[0](y) ← hangar[1]
4:      inner[1](x) ← hangar[2] − distance from plane origin to end of right wing
5:      inner[1](y) ← hangar[1]
6:      inner[2](x) ← inner[1](x)
7:      inner[2](y) ← hangar[3] − height of the plane
8:      inner[3](x) ← inner[0](x)
9:      inner[3](y) ← inner[2](y)
10: end function
```

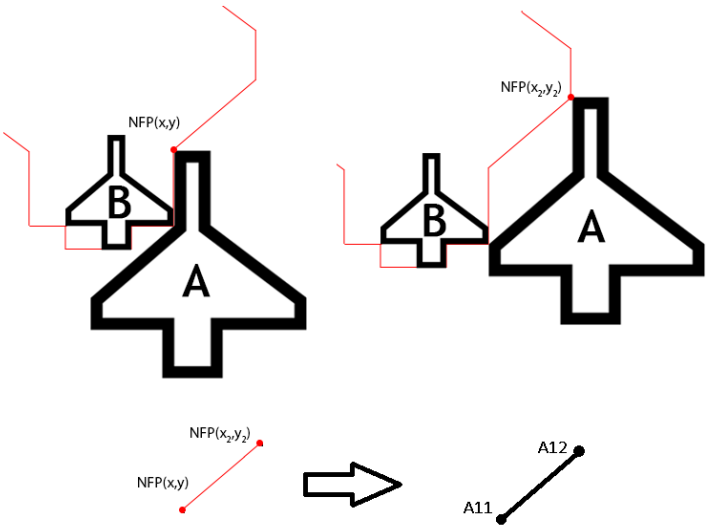

Figura 3.3: Exemplo de obtenção de ponto de NFP

3.3 Biblioteca de Rotinas de Suporte

Esta biblioteca recebe como entrada o conjunto de aviões a posicionar no hangar, pela ordem pela qual o seu posicionamento deve ser tentado. A partir desta lista ordenada de aviões, são calculados todos os NFP's possíveis entre os aviões recebidos. Para cada aeronave, são calculados todos os pontos possíveis de colocação, para que no final seja escolhido o que se encontre na posição mais abaixo e à esquerda do hangar. Usando o conceito de NFP, os pontos possíveis de colocação são os vértices dos NFP's do avião a colocar face aos já colocados, assim como as interseções entre as duas arestas, que não estejam no interior de um outro qualquer NFP ou fora do *innerfit*. Terminada a colocação é devolvido um ficheiro com a representação dos aviões colocados no hangar e outro com os dados para avaliação posterior. O procedimento descrito em Alg.4 descreve esta primeira parte de um ponto de vista geral.

Algorithm 4 Biblioteca de Rotinas de Suporte

```
1:  procedure BIBLIOTECA DE ROTINAS DE SUPORTE
2:      fp ← file with coordinates
3:      planes ← coordinates of every plane in fp
4:      generates nfp for each pair of planes
5:      for n ← 0 to n < nPlanes do
6:          calculates every feasible point to place the plane
7:          check if every listed point of placement is within the innerfit and outside of any NFP
8:          place plane in the most bottom-left point
9:      end for
10:     print SVG file
11:     save data from nesting
12: end procedure
```

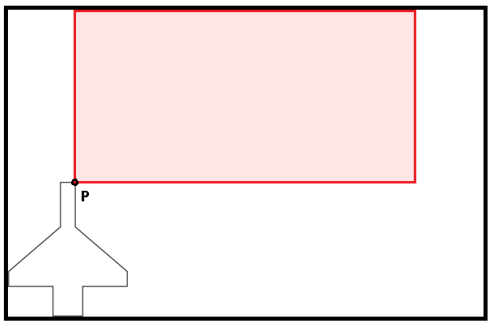

Figura 3.4: Innerfit polygon gerado por avião dentro do hangar

Inicialmente o programa recebe um ficheiro de texto .txt em que cada linha consiste nas coordenadas (x,y) dos vértices de cada aeronave e um valor final que representa a margem de segurança respetiva, tal como representado na Figura 3.6 .

A leitura e armazenamento das coordenadas de cada avião é realizada pelo algoritmo descrito no pseudocódigo Alg.5.

Tal como descrito em Alg.5, após a leitura do ficheiro, o algoritmo conta as linhas do ficheiro e o número de valores em cada linha. Cada linha é armazenada num *array* bidimensional do tipo *double*, planes[nplanes][points], que relaciona cada avião com os respetivos vértices.

De seguida é executada a função responsável por calcular todos os NFP's possíveis. Esta função é definida por noFitAB(plane_orb, plane_fix, nofit), e tem como argumentos os *arrays* de coordenadas do avião a orbitar e do avião fixo e o *array* bidimensional nofit_aux que retorna o cálculo da função. A função calcula os polígonos através de relações geométricas definidas a partir da forma geral dos NFP's obtida no estudo inicial 3.1. Ou seja, cada vértice do NFP resulta de uma relação com o vértice anterior ou com os vértices dos aviões fixo e orbitante. Em 3.1.1 é descrito em detalhe o processamento desta função.

Todos os NFP's gerados são gravados num *array* de 4 dimensões, $nofits[nPlanes][nPlanes-1][points][2]$, que relaciona dois aviões com o respetivo NFP para todas as combinações possíveis de aeronaves. Neste caso os *points* representam cada vértice do polígono compostos por um valor de x e de y cada.

Findo o cálculo de todos os NFP's, são verificadas as margens de segurança de cada avião em questão. A margem a utilizar será a maior entre ambos. É chamada a função $noFitMargin\ (nofits, margin)$ que expande o polígono calculado de acordo com a margem recebida. Estas margens não são totalmente realistas pois os cantos dos NFP's expandidos apresentam uma distância maior do que as arestas. Para obter uma expansão realista, os cantos deveriam ser curvos. Porém, usar curvas nas formas em questão, resulta numa dificuldade de cálculos geométricos acrescida, não sendo então consideradas. A Fig.3.7 demonstra a diferença entre a consideração ou não de margens de segurança entre os aviões, com os devidos NFP's representados. Como se pode ver em b), as margens são dadas pela expansão do NFP. O pseudocódigo Alg.6 descreve a expansão efetuada.

Algorithm 5 Armazenamento de coordenadas

```
1:  procedure ARMAZENAMENTO DE COORDENADAS
2:      fp ← FILE pointer to .txt
3:      junk ← character read from fp
4:      nPlanes ← number of lines read
5:      count ← 0
6:      points ← number of values in each line
7:      while junk ≠ EOF do
8:          if junk == newline character then
9:              nPlanes is incremented
10:             if points ≤ count then
11:                 points ← incremented count
12:             end if
13:             count ← 0
14:         else if junk == space character then
15:             count ←  is incremented
16:         end if
17:     end while
18:     go to: beginning of fp
19:     for i ← 0 to i < nPlanes do
20:         for j ← 0 to j < points do
21:             planes[i][j] ← value read from fp
22:         end for
23:     end for
24: end procedure
```

Algorithm 6 Expansão de NFP

```
1:  function NOFITMARGIN(NOFIT,MARGIN)
2:      for i ← 0 to i < 8 do
3:          nofit[i][0] ← nofit[i][0] + margin
4:          nofit[i][1] ← nofit[i][1] + margin
5:      end for
6:      for i ← 8 to i < 16 do
7:          nofit[i][0] ← nofit[i][0] + margin
8:          nofit[i][1] ← nofit[i][1] − margin
9:      end for
10:     for i ← 16 to i < 24 do
11:         nofit[i][0] ← nofit[i][0] − margin
12:         nofit[i][1] ← nofit[i][1] − margin
13:     end for
14:     for i ← 24 to i < 32 do
15:         nofit[i][0] ← nofit[i][0] − margin
16:         nofit[i][1] ← nofit[i][1] + margin
17:     end for
18: end function
```

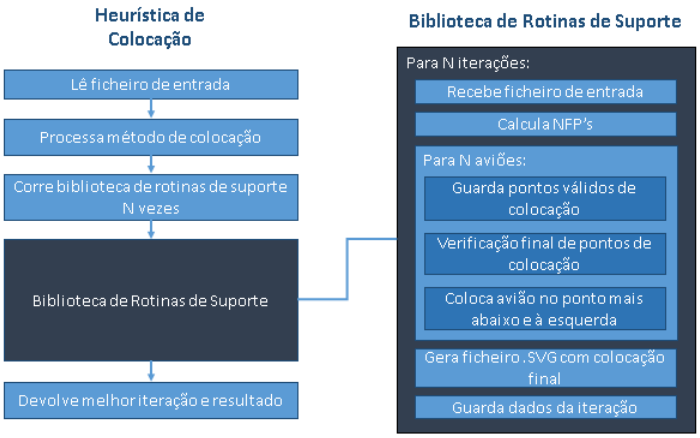

Figura 3.5: Diagrama da estrutura de funcionamento

A partir deste momento o programa vai guardar e por fim verificar, todos os pontos válidos de colocação para cada avião a posicionar. Os pontos válidos são os pontos que, verificando-se as condições de se encontrarem dentro do *innerfit* (3.1.2) e não se encontrarem dentro de NFP's, são considerados candidatos para a colocação do avião.

São fundamentais duas estruturas de armazenamento para tal. São estas os *arrays* val_points e valid_bool. O primeiro guarda cada coordenada a considerar e o segundo relaciona cada coordenada anterior com um valor de 1 ou 0, sendo que 1 representa um valor válido e 0 um valor a excluir. A escolha deste método deveu-se ao facto de não ser necessário a manipulação do *array* das coordenadas com inserções e exclusões, que de forma recorrente originariam erros.

Em primeiro é chamada a função innerFit(hangar,planes,inner) que a partir das coordenadas do hangar que se encontram previamente inicializadas e do avião a ler, devolve o *array* bidimensional *inner* para cada avião a colocar. O pseudocódigo Alg.3 descreve o procedimento do cálculo.

Visto que o algoritmo executa todos os cálculos e verificações a partir dos vértices dos aviões, NFP's, hangar e *innerfit*, todos os pontos válidos a considerar serão também vértices de NFP's, de *innerfit's* ou pontos resultantes de colisões entre estes. Assim sendo, os vértices do *innerfit* são guardados como pontos válidos de colocação, até que se verifique o contrário (encontrarem-se dentro de NFP's).

É verificado, a partir de uma variável inteira incrementada a cada colocação (*n_placed*), se já existe algum avião no hangar. Se não existir, o avião é imediatamente colocado no ponto válido com maior valor de y e menor de x, ou seja, o quarto vértice do *array inner*.

O próximo passo é adicionar a *val_points* todos os pontos dos NFP's a considerar, em que se verifique a condição de se encontrarem dentro do *innerfit*. Colocar um avião num ponto que se encontre dentro do *innerfit*, garante que este se encontra dentro do hangar (3.1.2). Um ponto

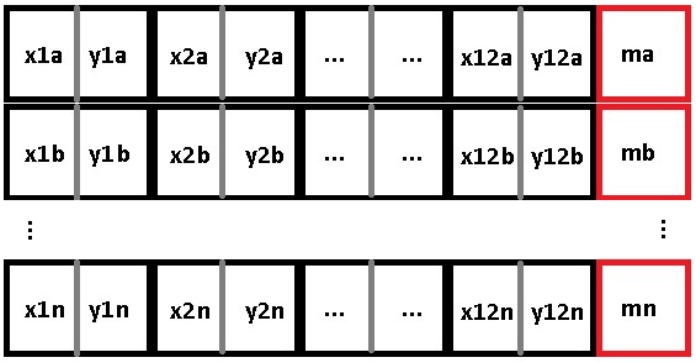

Figura 3.6: Esquema de ficheiro de entrada

$P(x,y)$ é considerado dentro do *innerfit* caso se verifique que:

$$\begin{aligned} innerfit_{x1} < P_x < innerfit_{x2} \\ innerfit_{y1} < P_y < innerfit_{y2} \end{aligned} \tag{3.1}$$

Relativamente aos pontos resultantes de colisões, estes são calculados em três etapas diferentes. Primeiramente as interseções entre NFP's e o *innerfit*, de seguida as colisões entre NFP's e por fim o caso em que um vértice de um NFP colide com um NFP.

Nos dois primeiros casos o cálculo é efetuado com recurso à função *intersection* 3.3.1, que recebe como parâmetros as coordenadas de dois segmentos de reta e o ponto resultante da colisão caso a função devolva o valor 1. Para cada avião colocado no hangar, é guardado num *array* auxiliar o NFP gerado entre este e o avião a colocar. De seguida, cada par de vértices do *innerfit* é guardado em dois *arrays* auxiliares para por fim, executar a verificação de interseção entre cada aresta que forma o NFP guardado e cada aresta do *innerfit*. O pseudocódigo 7 ilustra melhor o procedo desenvolvido.

De modo geral, são comparadas duas arestas de cada vez. Cada aresta do *innerfit* é comparada com todas as arestas de um NFP para verificar se existem interseções. Quando a variável *colision* assume o valor 1, o ponto de interseção calculado é guardado como ponto válido de colocação.

No segundo caso, o cálculo é bastante similar ao executado em 7, com a particularidade de que enquanto anteriormente os dois *arrays* auxiliares guardavam dois vértices seguidos do *innerfit*, agora guardam os vértices de um dos NFP's. Ao encontrar uma interseção o algoritmo verifica também se esta se encontra dentro do *innerfit*, caso em que o ponto não é considerado como válido para colocação.

O terceiro caso recorre à função pointInline 3.3.2 para verificar se um vértice se encontra sobre um segmento de reta. Verificando-se, a função devolve 1 e a colisão é o vértice em questão que é guardado, como anteriormente, caso este se encontre dentro do *innerfit*. O pseudocódigo para o algoritmo implementado é descrito em 9.

Algorithm 7 Colisão de NFP com innerfit

```
1: inner_points ← number of innerfit points
2: fitpoints ← number of points of one NFP
3: colision ← equals 1 if true or 0 if not
4: i ← plane to be placed
5: inter ← coordinates of intersection
6: for j ← 0 to j < n_placed do
7:     for j_inner ← 0 to j_inner < inner_points do
8:         aux ← inner[j_val]
9:         aux2 ← inner[j_val+1]
10:        for points ← 0 to points < fitpoints do
11:            colision ← intersection(aux,aux2, nofit[i][j][points], nofit[i][j][points+1], inter)
12:            if colision == 1 then
13:                inter stored as valid point
14:            end if
15:        end for
16:    end for
17: end for
```

Algorithm 8 Colisão entre dois NFP's

```
1: inner_points ← number of innerfit points
2: fitpoints ← number of points of one NFP
3: colision ← equals 1 if true or 0 if not
4: i ← plane to be placed
5: inter ← coordinates of intersection
6: for j ← 0 to j < n_placed do
7:     for j2 ← 0 to j2 < n_placed do
8:         if j2 ≠ j then                ▷ Ensure the NFP does not compare with self
9:             for points ← 0 to points < fitpoints do
10:                aux ← nofit[i][j][points]
11:                aux2 ← nofit[i][j][points+1]
12:                for points2 ← 0 to points2 < fitpoints do
13:                    colision ← intersection(aux,aux2, nofit[i][j2][points], nofit[i][j2][points+1], inter)
14:                    if (colision == 1)AND(interinsideinnerfit) then
15:                        inter stored as valid point
16:                    end if
17:                end for
18:            end for
19:        end if
20:    end for
21: end for
```

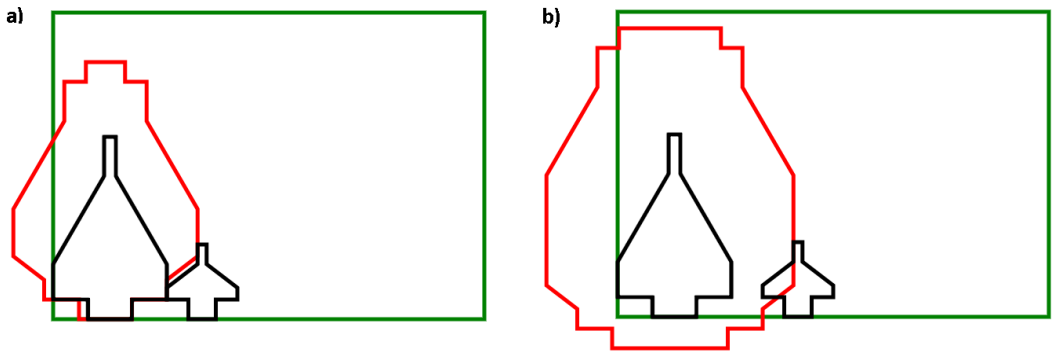

Figura 3.7: Colocação de aviões sem considerar margens de segurança em *a*) e com margens em *b*)

Para verificar se os pontos guardados como válidos se encontram dentro de qualquer um dos NFP's calculados inicialmente (o que resultaria numa colisão entre aviões), é chamada a função *inside(nofit, point)* que recebe um NFP e o ponto a examinar. Esta função delimita o NFP geral deduzido em retângulos e triângulos para a sua execução (Fig.3.8). A partir do momento em que o ponto se encontre numa destas formas, a função guarda numa variável um número inteiro maior que 0, que identifica a forma em questão. Caso contrário a função devolve 0.

Caso se encontre algum ponto dentro de um NFP, este é imediatamente marcado como inválido (valor 0 no *array valid_bool* correspondente ao ponto em questão).

Por fim, com todos os pontos válidos para colocação já obtidos, são procurados na lista os que têm o maior valor de y. Caso existam pontos marcados como válidos na lista, o que tiver a menor coordenada em x, será selecionado como ponto de colocação do vértice de origem do avião. As coordenadas do avião colocado são guardadas no *array placed*, incrementada a variável *n_placed* e zerada a lista de pontos válidos.

Finalizada a leitura de todas as aeronaves, o programa gera o ficheiro SVG com o desenho de todos os aviões colocados e o hangar. É ainda calculada e guardada num ficheiro de texto a área ocupada por todos os aviões no hangar para posterior análise da eficiência da colocação.

3.3.1 Função *intersection*

Para calcular o ponto de interseção entre dois segmentos de reta, que pode ser um ponto de colisão resultante entre dois NFP, foi necessário recorrer a uma função que, para tal, recebesse como argumento os pontos que delimitam os segmentos.

Usando um sistema de duas equações, seria fácil deduzir o ponto de interseção entre duas linhas. Porém, o que se pretende é o caso de dois segmentos de reta. A diferença reside no facto de que um reta é infinita ser infinita, ao contrário de um segmento de reta. Portanto, para além de ser necessário determinar onde as linhas que os segmentos definem se intersetam, é também necessário verificar se o ponto se encontra em ambos os segmentos. A fig.3.9, mostra como dois

Algorithm 9 Colisão de vértice de NFP com outro NFP

```
1:  inner_points ← number of innerfit points
2:  fitpoints ← number of points of one NFP
3:  colision ← equals 1 if true or 0 if not
4:  i ← plane to be placed
5:  for j ← 0 to j < n_placed do
6:      for j2 ← 0 to j2 < n_placed do
7:          if j2 ≠ j then                ▷ Ensure the NFP does not compare with self
8:              for points ← 0 to points < fitpoints do
9:                  aux ← nofit[i][j][points]
10:                 aux2 ← nofit[i][j][points+1]
11:                 for points2 ← 0 to points2 < fitpoints do
12:                     colision ← pointInline(aux,aux2, nofit[i][j2][points])
13:                     if (colision == 1)AND(nofit[i][j2][points]insideinnerfit) then
14:                         point stored as valid point
15:                     end if
16:                 end for
17:             end for
18:         end if
19:     end for
20: end for
```

segmentos de reta ($p_2 - p_1$ e $p_4 - p_3$) não se intersetam, ao contrário das linhas que os contêm que se intersetam num ponto p_i.

LaMothe (2002), descreve uma abordagem para este cálculo usando a representação paramétrica de cada segmento de reta a verificar. Para verificar se existe, e caso se confirme calcular, o ponto de interseção (x,y), definem-se os segmentos de reta como:

$$S_1 = p_1(x_1,y_1) - p_0(x_0,y_0) \tag{3.2}$$

$$S_2 = p_3(x_3,y_3) - p_2(x_2,y_2) \tag{3.3}$$

Nas seguintes equações, U representa o vetor posição de qualquer ponto em S_1 e V o vetor posição de qualquer ponto de S_2.

$$U = p_0 + t * S_1 \quad , \quad (0 <= t <= 1) \tag{3.4}$$

$$V = p_2 + s * S_2 \quad , \quad (0 <= s <= 1) \tag{3.5}$$

O que as equações 3.4 e 3.5 pretendem representar é o facto de que à medida que t e s variam desde 0 até 1, são percorridos os segmentos de p_0 a p_1 e p_2 a p_3 respetivamente. A figura 3.10 ilustra o referido.

É necessário agora resolver as equações 3.4 e 3.5 em ordem a s e t para, depois de obter estes

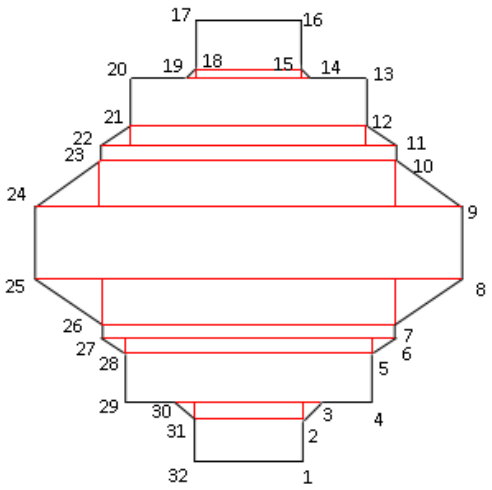

Figura 3.8: Delimitações das áreas da forma geral de NFP

valores, os substituir de volta nas equações. Obtendo as equações em ordem a s e t é também possível verificar que, caso alguma delas se encontre fora dos limites entre 0 e 1, não existe interseção entre os segmentos de reta.

Procede-se então à resolução das equações:

Para U = V, temos:

$$p_0 + t * S_1 = p_2 + s * S_2 \quad \Leftrightarrow \quad s * S_2 - t * S_1 = p_0 - p_2 \tag{3.6}$$

Como se pretende obter o valor das componentes (x,y),

$$s * S_2x - t * S_1x = p_0x - p_2x \tag{3.7}$$

$$s * S_2y - t * S_1y = p_0y - p_2y \tag{3.8}$$

Para resolver em ordem a (s,t) representamos as equações na forma $A * X = B$ através de matrizes, em que:

$$\text{A}=\begin{bmatrix} S_{2x} & -S_{1x} \\ S_{2y} & -S_{1y} \end{bmatrix}, \text{X}=\begin{bmatrix} s \\ t \end{bmatrix} \text{ e B}=\begin{bmatrix} p_{0x} & -p_{2x} \\ p_{0y} & -p_{2y} \end{bmatrix}$$

Tal como é detalhado em profundidade por Werner (1984), a partir da regra de Cramer, podemos solucionar uma equação na forma $A * X = B$ para obter x, sendo neste caso os valores de s e

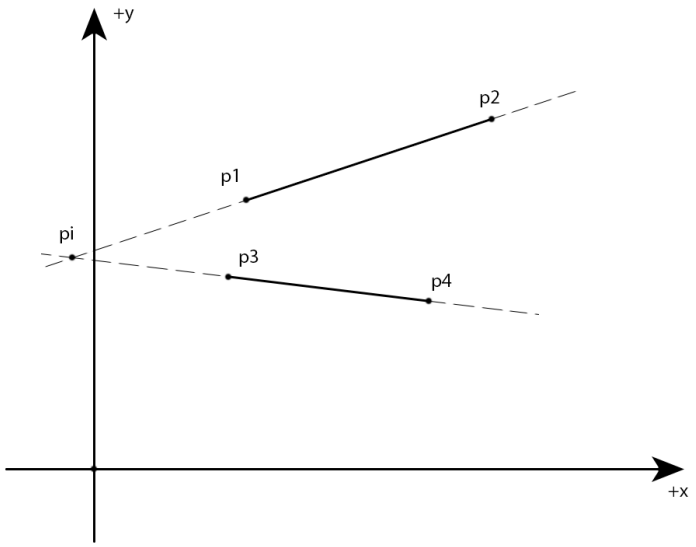

Figura 3.9: Exemplo de diferença entre reta e segmento de reta

t. O teorema afirma que os valores desconhecidos a determinar são dados por:

$$x_i = \frac{det(A_i)}{det(A)} \tag{3.9}$$

Em que,

x_i é a matriz de variáveis a determinar, sendo neste caso $x_1 = s$ e $x_2 = t$;

det representa o cálculo do determinante de uma matriz;

A_i é a matriz que resulta da substituição da coluna i de A com B.

Tem-se então:

$$s = \frac{det\begin{vmatrix} (p_{0x} - p_{2x}) & -S_{1x} \\ (p_{0y} - p_{2y}) & -S_{1y} \end{vmatrix}}{det\begin{vmatrix} S_{2x} & -S_{1x} \\ S_{2y} & -S_{1y} \end{vmatrix}} \tag{3.10}$$

$$t = \frac{det\begin{vmatrix} S_{2x} & (p_{0x} - p_{2x}) \\ S_{2y} & (p_{0y} - p_{2y}) \end{vmatrix}}{det\begin{vmatrix} S_{2x} & -S_{1x} \\ S_{2y} & -S_{1y} \end{vmatrix}} \tag{3.11}$$

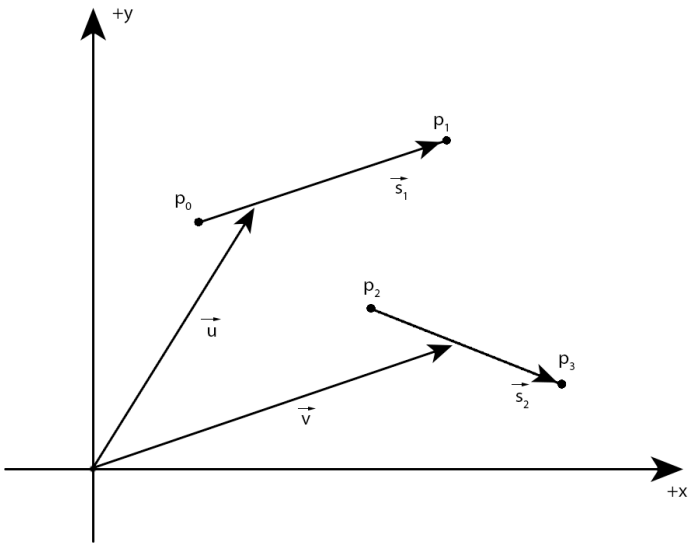

Figura 3.10: Representação paramétrica de U e V

O determinante de uma matriz 2x2 é bastante fácil de calcular, sendo o resultado da multiplicação do primeiro elemento com o quarto a subtrair pela multiplicação do segundo com o terceiro elemento. No final, o cálculo das equações 3.10 e 3.11 resulta respetivamente em:

$$s = \frac{-S_{1y} * (p_{0x} - p_{2x}) + S_{1x} * (p_{0y} - p_{2y})}{-S_{2x} * S_{1y} + S_{1x} * S_{2y}} \tag{3.12}$$

$$t = \frac{S_{2x} * (p_{0y} - p_{2y}) - S_{2y} * (p_{0x} - p_{2x})}{-S_{2x} * S_{1y} + S_{1x} * S_{2y}} \tag{3.13}$$

Substituindo os valores de s e t nas equações 3.4 e 3.5 e caso se verifiquem as condições destas, obtêm-se os valores (x,y) do ponto de interseção.

3.3.2 Função pointInline

A lógica fundamental desta função, assenta no cálculo de distâncias entre pontos. Ou seja, para verificar se um ponto P se encontra sobre o segmento de reta dado por dois pontos A e B, são determinadas as distâncias AP, PB e AB. Se a soma de AP com PB igualar AB, então o ponto encontra-se sobre o segmento. Na Figura 3.11 podemos verificar a diferença entre o caso em que a função devolve 1 e quando devolve 0.

3.3.3 Comparação entre valores do tipo *double*

Quando se executam cálculos com valores do tipo *float* ou *double*, uma comparação com '==' ou'!=' é insegura. Ao comparar o valor 0.1 com $float(0.1)$ ou $double(0.1)$ observa-se que

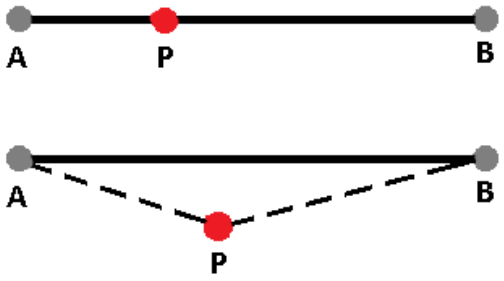

Figura 3.11: Posição de ponto *P* relativa a um segmento *AB*

são distintos (Tabela 3.1). Isto deve-se ao facto de, tanto $float(0.1)$ como $double(0.1)$, serem arredondamentos de 0.1 e não o valor exato que se pretende.

Representação	Valor
0.1	0.1
float(0.1)	0.100000001490116119384765625
double(0.1)	0.1000000000000000055511151231257827021181583404541015625

Tabela 3.1: Valores de 0.1

Assim, recorre-se a uma função que a partir da diferença entre os dois valores a comparar, averigua se esta se encontra dentro de certos limites de erro ou valor de *epsilon*. Sendo verdadeiro, os valores são próximos o suficiente para se considerarem iguais (Dawson (2012)). O algoritmo implementado é descrito pelo pseudocódigo Alg.10.

3.4 Heurística de Colocação

Encontrando-se completa a camada que processa todos os cálculos e respetiva colocação dos aviões, é necessário que este processo seja executado várias vezes para se chegar a várias soluções possíveis e, a partir dos resultados obtidos, devolver a iteração que apresente a solução mais viável.

O programa lê então o ficheiro que contém as aeronaves a colocar e procede à escolha e processamento do método escolhido. A leitura é executada tal como descrito anteriormente no 5. A partir do momento em que o método é escolhido, este vai escrever um novo ficheiro a enviar para a biblioteca de rotinas de suporte que essencialmente consiste numa reordenação dos aviões. Isto resulta, pois a biblioteca de rotinas de suporte executa a colocação por ordem de leitura de cada avião. Tem-se então a possibilidade de ordenação aleatória (*shuffle*) e por tamanho (crescente ou decrescente). Note-se que na segunda alternativa, torna-se supérfluo o uso de várias iterações, pois a ordem será sempre a mesma.

Algorithm 10 Função de comparação de valores *double*

```
1:  function ALMOSTEQUAL(A,B,EPSILON)
2:      diff ← |A − B|
3:      A ← |A|
4:      B ← |B|
5:      if B > A then
6:          largest ← B
7:      else if A > B then
8:          largest ← A
9:      end if
10:     if diff ≤ (largest ∗ epsilon) then
          return true
11:     else
          return false
12:     end if
13: end function
```

Após se obter o número de iterações pretendidas, o programa executa em ciclo a chamada do ficheiro *.exe* que contém a biblioteca de rotinas de suporte e passa como argumento o novo ficheiro de entrada. Esta solução deve-se ao facto de que, quando chamamos a biblioteca de rotinas de suporte através da função *system()* incluída na biblioteca *windows.h*, no final de cada ciclo as variáveis usadas são limpas da memória, evitando-se assim possíveis problemas.

Finalizado o ciclo, o programa lê o ficheiro que contém os dados de cada iteração para devolver qual delas foi a melhor e o respetivo resultado. O processo responsável pela heurística de colocação é explicado pelo pseudocódigo 11.

Algorithm 11 Heurística de Colocação

```
1:  procedure HEURÍSTICA DE COLOCAÇÃO
2:      fp ← file with coordinates
3:      planes ← coordinates of every plane in fp
4:      for n ← 0 to n < cycles do
5:          if method == s then
6:              shuffle(planes,nPlanes)
7:          else if method == a then
8:              sizeSortAsc(planes,nPlanes)
9:          else if method == d then
10:             sizeSortDesc(planes,nPlanes)
11:         end if
12:         fw ← rearranged planes order file
13:         system(nofit.exe − fw)          ▷ Calls .exe containing lower layer
14:     end for
15:     read data.txt file
16:     print best result
17: end procedure
```

3.4.1 Ordenação por shuffle

Para a ordenação aleatória, desenvolveu-se uma função *shuffle(planes[], size)* que recebe o *array* com todos os aviões pela ordem de leitura do ficheiro original e devolve o *array* com as posições baralhadas. O novo ficheiro de entrada gerado, abordado anteriormente, é escrito dentro do ciclo onde ocorre a chamada da função *shuffle*.

A execução deste método passa pela utilização das funções *rand()*, que retorna um número pseudo-aleatório entre 0 e *RAND_MAX*, e *srand()* que gera a semente a utilizar por *rand()*. Como os números gerados não são verdadeiramente aleatórios, se a semente se manter inalterada, cada execução do programa irá produzir os mesmos resultados. Apesar de ser uma desvantagem por não ser realmente aleatório, a grande vantagem é que desta maneira, a obtenção da mesma sequência exata, é importante para a deteção e correção de erros.

Outra particularidade é o facto de que a semente deve ser "aquecida"previamente, caso contrário a primeira posição seria sempre a mesma. O algoritmo que gera os números aleatórios, conhecido por Método congruencial multiplicativo (Oliveira e Carravilla (2010)), é dado por:

$$n_{i+1} = (b * n_i + c) mod(m) \tag{3.14}$$

Em que:

m é o maior número que pode ser gerado (*RAND_MAX*),

b e c são parâmetros da formula,

e o primeiro valor de n (n_0) é a semente do gerador.

O função é descrita pelo pseudocódigo 12.

Algorithm 12 Função *Shuffle*

```
1:  function SHUFFLE(PLANES,NPLANES)
2:      srand(seed)
3:      if nPlanes > 1 then
4:          for x ← 0 to x < 1000 do
5:              rand()                                  ▷ 'Warm' seed
6:          end for
7:          for i ← n − 1 to i > 0 do
8:              j ← rand()%(i + 1)
9:              aux ← planes(j)
10:             planes(j) ← planes(i)
11:             planes(i) ← aux
12:         end for
13:     end if
14: end function
```

3.4.2 Ordenação por tamanho

Esta ordenação é realizada pelas funções *sizeSortDesc(planes[], size)* ou *sizeSortAsc(planes[], size)*, dependendo se se pretende uma ordenação por tamanho do avião maior para o mais pequeno ou do avião mais pequeno para o maior, respetivamente. A função auxiliar *planeArea(plane[])* calcula a área de cada avião para se proceder ao processamento do método. O pseudocódigo da função que ordena por ordem decrescente é descrito em 13. Para a função que ordena por ordem crescente, o código é exatamente o mesmo com a simples alteração do sinal de comparação.

Algorithm 13 Função de ordenação decrescente

```
1:  function SIZESORTDESC(PLANES,NPLANES)
2:      for i ← 0 to i < nPlanes do
3:          for j ← i+1 to j < nPlanes do
4:              if then planeArea(planes(i)) < planeArea(planes(j))   ▷ switch '<' with '>' for ascending order
5:                  aux ← planes(j)
6:                  planes(j) ← planes(i)
7:                  planes(i) ← aux
8:              end if
9:          end for
10:     end for
11: end function
```

Capítulo 4

Resultados

Neste capítulo são descritas todas as parametrizações consideradas e as respetivas simulações executadas com base nesses dados.

Estes testes foram efetuados com recurso a um computador com processador Intel(R) Core (TM) i7-4510U de 2,00GHz e 8,00 GB de RAM sobre o sistema operativo Windows 10 Home. O código foi totalmente escrito em C e compilado no programa CodeBlocks v13.12.

4.1 Instâncias

Para a realização dos testes computacionais foram considerados os dados usados por QIN et al. (2017) para estudo realizado no mesmo âmbito. Estes dados dizem respeito a um hangar em Hong Kong com as dimensões de 110 metros de largura e 110 metros de altura. Esta informação foi recolhida entre Janeiro e Maio de 2015. Atualmente, o fornecedor do serviço de manutenção em questão, planeia as colocações dos aviões manualmente com períodos regulares. Os diferentes aviões considerados são enumerados na tabela 4.1.

Categoria	Modelos de Aviões	Margens
Aviões de pequena dimensão ($< 300m^2$)	G200; CL600; CL605; F900LX; F2000EX; F2000LX; ERJ135; F7X; G450; GIV	3m
Aviões de média dimensão ($300m^2 <= \text{dimensão} < 1500m^2$)	GL5T; G550; G5000; G6000; G650; A318; ERJ190; A319; A320; B738; A321	4m
Aviões de grande dimensão ($< 1500m^2$)	A332; A333	5m

Tabela 4.1: Modelos de aviões e respetivas margens de segurança

A partir destas dimensões determinaram-se para cada um destes três tipos de aeronaves as suas margens de segurança. As margens definidas são de três metros, quatro metros e cinco metros para os aviões pequenos, médios e grandes respetivamente.

A abordagem feita considera como resultados da colocação, para posterior comparação, a proporção de espaço ocupado sobre o espaço livre no hangar. É também admitido que o máximo lucro advém da operação de manutenção de aviões de maior dimensão. Assim sendo o valor de cada um é dado pelo cálculo da sua área. Visto que os maiores aviões são os que transportam um maior número de pessoas ou mercadorias e os que necessitam de maior tempo de manutenção, ao efetuar a colocação dos aeronaves pelo seu tamanho, assume-se que o subconjunto resultante é o dos aviões de maior lucro para a empresa.

4.2 Testes Computacionais

A tabela 4.2 apresenta os resultados dos testes computacionais efetuadas para o método de colocação de aviões aleatório e por tamanho crescente. Para cada instância foram executadas 10 iterações e os resultados apresentados correspondem à iteração com a percentagem de utilização do hangar de maior valor. No caso da ordenação aleatória é ainda apresentada a média e o desvio padrão relativos ao número de aviões colocados nas várias iterações de cada instância. As duas últimas colunas apresentam os valores obtidos por QIN et al. (2017) ao simular as mesmas instâncias. O tempo médio que o programa demora a devolver os cálculos das 10 iterações encontra-se entre os 2 a 3 segundos.

Instância	Aviões a colocar	Ordenação aleatória					Ordenação por tamanho		Resultados de QIN et al. (2017)	
		Aviões colocados			Percentagem de aviões colocados	Utilização do hangar (%)	Aviões colocados	Utilização do hangar (%)	Aviões colocados	Utilização do hangar (%)
		Média	Des. Padrão	Melhor						
1	6	6	0	6	100	28.896	6	28.896	6	29.0
2	6	5.7	0.458	6	100	30.058	6	30.058	6	30.0
3	7	6.7	0.458	7	100	35.009	6	21.916	7	35.0
4	7	5.5	0.5	6	85.7	36.478	5	23.384	7	39.0
5	8	7.7	0.458	8	100	32.330	7	20.592	8	32.0
6	8	6.8	0.4	7	87.5	30.948	6	17.855	8	44.0
7	9	8.3	0.458	9	100	33.761	7	18.393	9	34.0
8	10	9	0.447	10	100	37.604	8	18.443	10	38.0
9	11	10.2	0.6	11	100	33.924	10	31.649	11	33.0
10	12	9.6	0.489	10	83.3	33.822	10	30.589	12	46.0

Tabela 4.2: Resultados dos testes computacionais

Capítulo 5

Conclusões e Trabalho Futuro

A realização desta dissertação teve como principal motivação o desenvolvimento de uma abordagem mais simples e eficiente para o problema de colocação de aviões num hangar, tendo em conta certas considerações como as margens de segurança entre aviões. Para tal, foram estudadas as ferramentas geométricas existentes e de que maneira podiam ser utilizadas na resolução deste problema. Concluiu-se que a complexidade dos algoritmos existentes para a construção de NFP's ia contra a simplicidade que se pretendia. A partir do estudo dos NFP's gerados entre aviões de diferentes dimensões, observou-se que existiam características comuns a todos eles, o que permitiu criar um algoritmo inovador que relaciona matematicamente os vértices dos aviões com o NFP gerado por eles.

5.1 Satisfação dos Objetivos

A partir dos resultados dos testes computacionais efetuados é possível retirar algumas ilações. Na tabela Tab.4.2, verifica-se que a ordenação realizada através do tamanho dos aviões (do mais pequenos para o maior) apresenta resultados menos favoráveis do que a ordenação aleatória, quando o número de aviões colocados é efetivamente menor do que os que se pretende colocar. Em certos casos como as instâncias 7 e 8, o número de aviões colocados por ordem de tamanho é consideravelmente menor do que o melhor caso aleatório, tendo um impacto enorme na eficiência deste método. Em comparação com os resultados obtidos por QIN et al. (2017), é de notar que quando os aviões de entrada não são todos colocados, a percentagem de utilização do hangar nalgumas das instâncias fica um pouco abaixo dos valores que estes obtiveram nas suas simulações. Porém, a maioria encontra-se próxima dos resultados ideais. Uma possível explicação para esse facto encontra-se no facto do número de iterações utilizadas em cada instâncias não ser o suficiente, visto que quanto mais iterações se realizarem, maior será a probabilidade de obter a ordem ideal de entrada dos aviões.

5.2 Trabalho Futuro

O trabalho descrito nesta dissertação seria amplamente beneficiado com alguns melhoramentos num trabalho futuro.

Um dos aspetos que seria interessante abordar está relacionado com a rotação dos aviões. Desenvolver um algoritmo que permita a colocação das aeronaves em diferentes ângulos, facilitaria por exemplo as manobras de entrada e saída do hangar.

Relativamente à implementação do algoritmo, no futuro o algoritmo pode ser reescrito utilizando uma linguagem orientada a objetos. Desta maneira, as estruturas que compõem os aviões e NFP's serão mais fáceis de implementar e modificar. A construção do NFP pode também beneficiar com isto, passando a serem feitas relações entre as arestas que compõem os aviões em vez do uso dos seus vértices.

No que diz respeito à utilização do espaço, a colocação dos aviões pode ser efetuada num espaço tridimensional. Deste modo, pode ser considerada a colocação de aeronaves pequenas debaixo das asas de aeronaves maiores, por exemplo.

Os métodos de colocação podem nunca resultar numa solução ótima pelo que diferentes métodos heurísticos devem ser considerados no futuro.

Anexo A

Resultados Computacionais

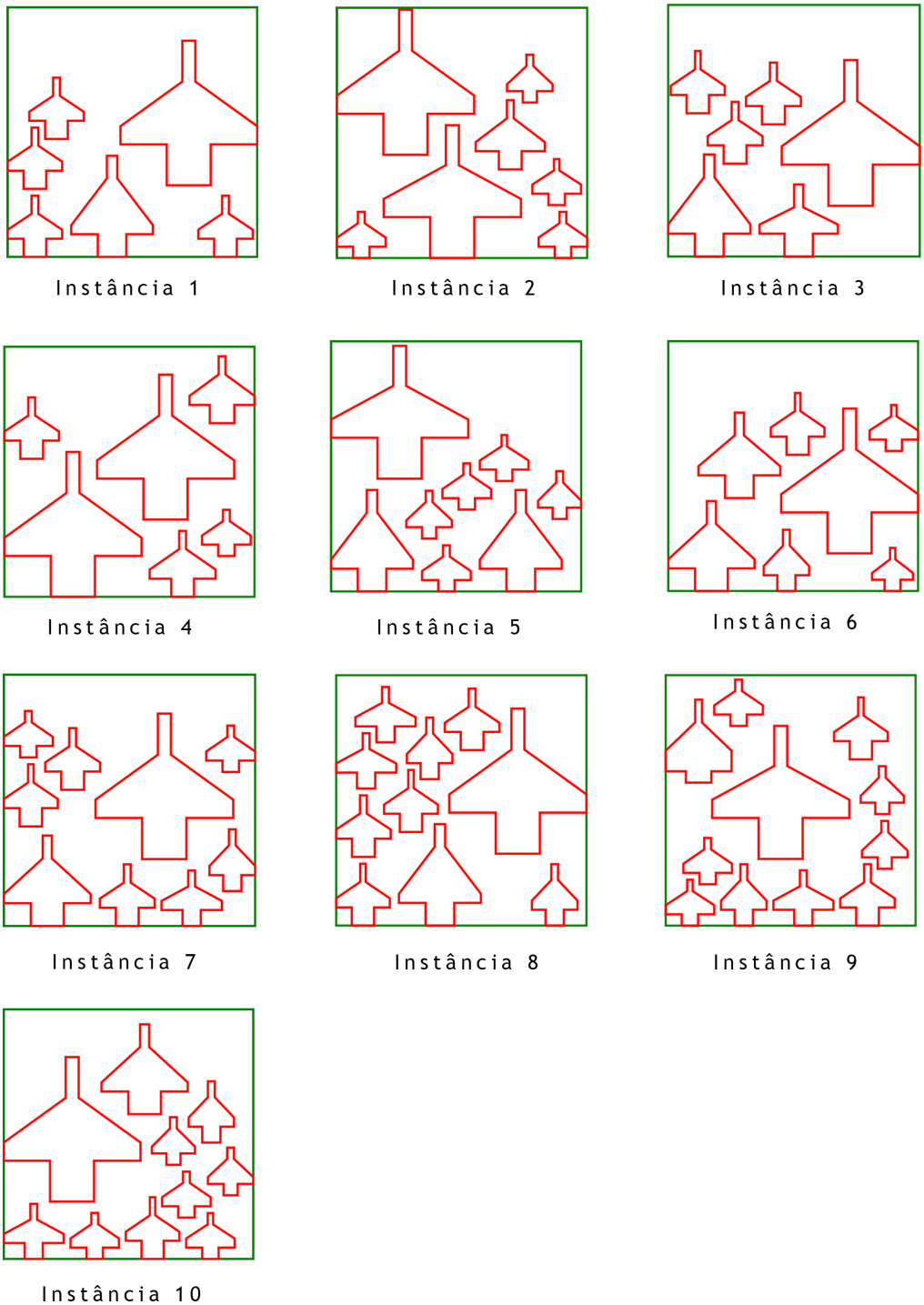

Figura A.1: Melhores resultados computacionais obtidos para cada instância

Referências

Pankaj K. Agarwal, Eyal Flato e Dan Halperin. Polygon decomposition for efficient construction of Minkowski sums. *Computational Geometry*, 21(1–2):39–61, 2002. ISSN 0925-7721. doi: 10.1016/S0925-7721(01)00041-4.

Francis Avnaim e J Bsissonnat. Simultaneous containment of several polygons. *Proceedings of the third annual symposium on Computational geometry - SCG '87*, páginas 242–247, 1987. doi: 10.1145/41958.41984.

Julia A. Bennell e Jose F. Oliveira. The geometry of nesting problems: A tutorial. *European Journal of Operational Research*, 184(2):397–415, 2008. ISSN 03772217. doi: 10.1016/j.ejor.2006.11.038.

Julia A. Bennell e Xiang Song. A comprehensive and robust procedure for obtaining the nofit polygon using Minkowski sums. *Computers and Operations Research*, 35(1):267–281, 2008. ISSN 03050548. doi: 10.1016/j.cor.2006.02.026.

Julia A. Bennell, Kathryn A. Dowsland e William B. Dowsland. The irregular cutting-stock problem - a new procedure for deriving the no-fit polygon. *Computers and Operations Research*, 28(3):271–287, 2000. ISSN 03050548. doi: 10.1016/S0305-0548(00)00021-6.

E. K. Burke, R. S R Hellier, G. Kendall e G. Whitwell. Complete and robust no-fit polygon generation for the irregular stock cutting problem. *European Journal of Operational Research*, 179(1):27–49, 2007. ISSN 03772217. doi: 10.1016/j.ejor.2006.03.011.

R Cuninghame-Green. Geometry, shoemaking and the milk tray problem. *New Scientist 12th August 1989*, 1677:50–53, 1989.

Ray Cuninghame-Green. Cut out waste! *OR Insight*, 5(3):4–7, Jul 1992. ISSN 1759-0477. doi: 10.1057/ori.1992.20.

Bruce Dawson. Comparing floating point numbers. https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition, 2012. Acedido em : 12-11-2017.

J. Fernández, L. Cánovas e B. Pelegrín. Algorithms for the decomposition of a polygon into convex polygons. *European Journal of Operational Research*, 121(2):330–342, 2000. ISSN 03772217. doi: 10.1016/S0377-2217(99)00033-8.

Pijush K. Ghosh. An algebra of polygons through the notion of negative shapes. *CVGIP: Image Understanding*, 54(1):119–144, 1991. ISSN 10499660. doi: 10.1016/1049-9660(91)90078-4.

M Konopasek. Mathematical treatment of some apparel marking and cutting problems. *US Department of Commerce Report, 99-26-90857-10*, 1981.

Andre LaMothe. *Tricks of the Windows Game Programming Gurus*. Sams, Indianapolis, IN, USA, 2nd edição, 2002. ISBN 0672323699.

Zhenyu Li e Victor Milenkovic. Compaction and separation algorithms for non-convex polygons and their applications. *European Journal of Operational Research*, 84(3):539–561, 1995. ISSN 03772217. doi: 10.1016/0377-2217(95)00021-H.

A Mahadevan. Optimisation in computer aided pattern packing. *Ph.D Thesis, North Carolina State University*, 1984.

José Oliveira e Maria Carravilla. Simulação - transparências de apoio à lecionação de aulas teóricas. *Faculdade de Engenharia da Universidade do Porto*, páginas 13–16, 2010.

Brian Pearce. Economic performance of the airline industry mid-year report. https://www.iata.org/whatwedo/Documents/economics/IATA-Economic-Performance-of-the-Industry-mid-year-2017-report.pdf, 2017. Acedido em : 05-01-2018.

Yichen QIN, Felix T. S. CHAN, S.H. CHUNG, T. QU e B. NIU. Aircraft parking stand allocation problem with safety consideration for independent hangar maintenance service providers. *Computers and Operations Research*, 2017. doi: 10.1016/j.cor.2017.10.001.

Raimund Seidel. Reprint of: A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, 43(6-7):556–564, 2010. ISSN 09257721. doi: 10.1016/j.comgeo.2010.03.003.

Y. Stoyan e L.D. Ponomarenko. Minkowski sum and hodograph of the dense placement vector function. *Reports of the SSR Academy of Science, SER. A*, 10, 1977.

Yurij Stoyan, Guntram Scheithauer, Nikolay Gil e Tatiana Romanova. Φ-functions for complex 2D-objects. *4or*, 2(1):69–84, 2004. ISSN 16142411. doi: 10.1007/s10288-003-0027-1.

Jorne Van den Bergh, Philippe De Bruecker, Jeroen Beliën e Jonas Peeters. Aircraft maintenance operations: state of the art. *Faculteit Economie En Bedrijfswetenschappen*, página 34, 2013.

Hans Joachim Werner. On extensions of Cramer's rule for solutions of restricted linear systems. *Linear and Multilinear Algebra*, 15(3-4):319–330, 1984. ISSN 0308-1087. doi: 10.1080/03081088408817600.